Ethernet Interface Documentation Package

John Seamons and Andreas Bechtolsheim
Computer Science Department
Stanford University

June 30, 1980
July 3, 1980 (old)

This set of documents describes a 3 Megabit Ethernet Interface developed by the Computer Science Department at Stanford University. The interface is designed to interconnect host computers to the Ethernet via a 16-bit parallel port.

The package contains:

* Function and Interface Description
* Logic Schematics
* Prototype Layout
* Timing Diagrams
* State Machine Graphs
* State Machine Code

```
COMMENT ⊛   VALID 00007 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    Ethernet Interface Documentation
C00005 00003    Ethernet Receiver
C00008 00004    Ethernet Transmitter
C00010 00005    Ethernet Controller
C00012 00006    DMA Controller:
C00015 00007    Controller Function:
C00018 ENDMK
C⊛;
```

Ethernet Interface Documentation

The Ethernet interface described in the following connects
an Ethernet transceiver to one of three system interfaces:
1) A 8/16 bits parallel port.
2) The GPIB General Purpose Interface Bus.
3) A RS-422 high-speed serial line.
The interface is designed for full-duplex operation.
Thus loopback (sending to oneself) is possible for diagnostic purposes.

The Ethernet interface itself consists of three parts:
a Receiver, a Transmitter, and a Controller.

The receiver converts serial data from the Ethernet into 16-bit parallel words.
A Bit-vector address filter, which is loaded from the controller,
accepts only a selected set of destination addresses.
The receiver also straps the Ethernet leader bit and checks the CRC.

The transmitter translates 16-bit parallel data into a serial data stream.
It prefixes the data stream with a leader bit and appends the CRC code
which it computes on the fly.
In addition, the transmitter jams the Ethernet if a collision is detected
either by the transceiver or the receiver.

The controller performs the other functions necessary for a system level
Ethernet interface: packet buffering, retransmission, and initialization.
The controller interfaces the Ethernet to either a 8/16 bit parallel port,
a GPIB port, or a high-speed serial line port.
These ports can be easily interfaced to almost any host computer.
The controller is implemented as a fast 8-bit microcomputer system with
four DMA channels, Timer, and 4 kBytes of buffer memory.

If desired, the controller's functionality can be replaced by a
"host" mini- or microcomputer system. However, the timing characteristics
of the Ethernet (one 16-bit word every 5.44 usec, retransmission within 37 usec)
need to be carefully considered for alternative controller implementations.

Ethernet Receiver

Receiver Interface:

R.Data <0:15>	Output	Parallel Data.
R.Read0\	Input	Read Byte 0 (Bits <0:7>). } Tie together for
R.Read1\	Input	Read Byte 1 (Bits <8:15>). } 16 bit operation.
R.Read\	Input	Clear flags. Tie to R.Read0/1.
R.Ready\	Output	New data ready.
R.End\	Output	End of Message. Cleared by R.Ack.
R.Abort\	Output	Abort message in progress. Cleared by R.Ack.
R.Ack\	Input	Clear R.End and R.Abort.
R.Enable	Input	Initializes receiver state machine. Assert after initialization is complete.
R.FA <0:7>	In/Out	Filter Address. Shift Register Output if R.Enable is asserted.
R.FDin	Input	Filter Data In
R.FWE\	Input	Filter Write Enable

Receiver Functions:

If R.Enable non asserted, then receiver state machine is initialized.
Address filter can be loaded via R.FA, R.FDin, and R.FWE\.
Address filter should be loaded before interface is enabled.

After R.Enable is asserted, receiver state machine will enter
flush state until carrier idle.
Receiver then listens continously to Ethernet.
Phase Decoder acquires synchronization on Start Bit.
Receiver state machine places data into shift register.
At the end of the first word it checks the address in the address vector.
If Ethernet carrier drops, receive state machine checks
CRC and Bit-count. If both are correct, R.End flag is set.
Otherwise and in the case of collision after the first R.Ready
R.Abort is set.
Microcomputer system should throw away aborted messages.
(Number of aborted messages is expected to be very small.)

Response time requirements:

R.Ready to R.Read:	5440 nsec (16 bit times).
R.End to R.Ready:	6460 nsec (19 bit times).

Ethernet Transmitter

Transmitter Interface:

T.Data <0:15>	Input	Parallel Data.
T.Write0\	Input	Write Byte 0 (Bits <0:7>). } Tie together for
T.Write1\	Input	Write Byte 1 (Bits <8:15>). } 16 bit operation.
T.Write\	Input	Clear flags. Tie to Write0/1.
T.Ready\	Output	Ready to accept data. Cleared by T.Write.
T.Sent\	Output	Message transmitted. Cleared by T.Ack.
T.Abort\	Output	Message collided. Cleared by T.Ack.
T.Ack\	Input	Clear T.Sent and T.Abort.
T.Enable	Input	Enable transmit data driver to transceiver. Can be used to terminate transmissions immediately. Should be held inactive after power-up to allow transmitter state machine to autoinitialize.

Transmitter Functions:

Transmitter is activated by writing first data word.
Transmitter then attempts to aquire Ethernet.
Timeout function for aquisition must be provided externally.
Transmission is terminated with CRC code if no more data is available.
Transmission is aborted upon collision reported by Transceiver or Phase Decoder.
Retransmission function and algorithm must be provided externally.

Response time requirements:

T.Write to T.Ready:	5440 nsec (16 bit times).
T.Abort to T.Write:	37 usec (1 retransmission slot time).

Ethernet Controller

Controller Interface:

The controller supports one of three bidirectional data ports:
a parallel port, a GPIB port, and a SIO port.
For a particular system, only one of these ports can be used.
The particular port in use will be called "local port".

The interface is implemented via messages consisting of one 16-bit
command word that is optionally followed by a fixed number of data words.
The length of the data field is limited to 512 words (1024 bytes).

Message Summary:

Host to Ethernet_Controller:

Initialize
Load Address Filter with <n> address
What is your fixed address?
Send following packet with <n> words, <words>*n

Ethernet Controller to Host:

My fixed address is <n>
Timeout on network aquisition
I failed to send a packet despite 8 retries
I received following packet with <n> words, <words>*n

DMA Controller:

All data transfers are handled by the 9517 DMA Controller.
This DMA chip has four channels allocated as follows:

Channel 0 Receiver
Channel 1 Receiver EOP Pointer
Channel 2 Transmitter
Channel 3 Local Port

Chan 0: Autoinitialization implements cyclical buffer.
Save all receiver data in buffer (including bad messages)
Chan 1: Autoinitialization implements cyclical buffer.
Save current address and R.Abort on R.End.
Chan 2: Transfer fixed number of data words from buffer to transmitter.
Chan 3: Transfer fixed number of data words between local port and buffer.

The first two channels allow to receive back-to-back packets while
the buffer memory can be simultaneously unloaded to the local port.

Both the channel for Receiver Data and EOP Address perform their task
without intervention of the microprocessor and deposit their data items
into cyclical queues in buffer memory.
In contrast, the local port Read/Write and Transmitter Data channel
are explicitely started under microcomputer control with a
definite wordcount, which is the length of the packet to be transferred.
The Microcomputer is interrupted on EOP of a local port transfer
and on SENT from the transmitter. Note that SENT is set whenever the
transmitter terminated its transmission, which might be on
collision and on data underflow in addition to regular completion.

Controller Function:

Upon initialization (either hardware or command), the 8085 controller:
1) reads the fixed address and loads it into the address filter,
2) initializes the Ports
3) initializes the DMA device
4) enables the receiver and the transmitter
5) enables interrupts
6) halts.
All further functions are interrupt driven (described below).

Interrupt routines:

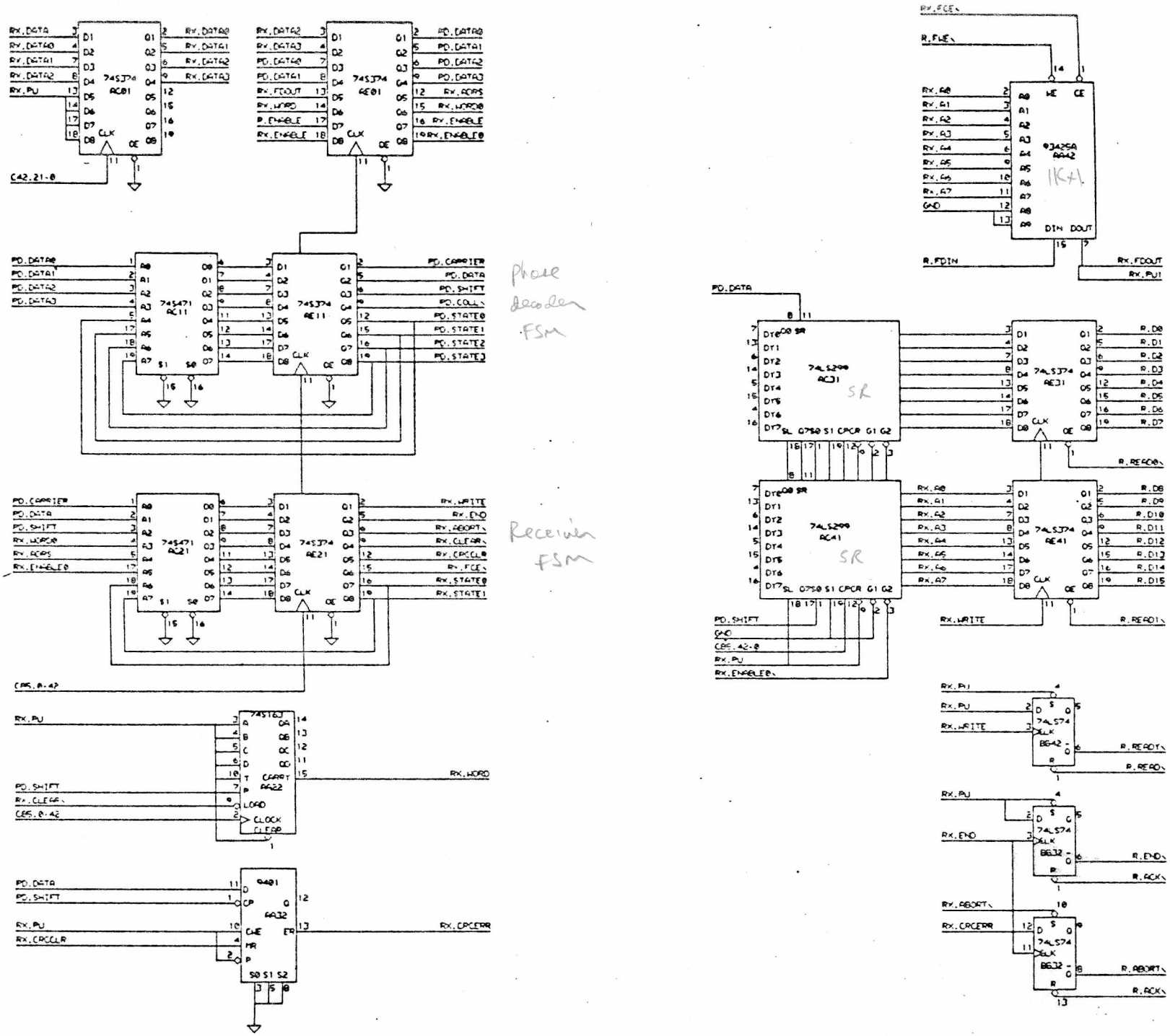
Trap (edge and level, non-maskable)
not used because of difficulty to synchronize DMA device manipulation.

RST 7.5 (edge level)
TO (Timer: Count=0)
Timer is started on network aquisition for timeout and
on collision for retransmission.
At other times timer is running continously for random number generation.
RST 7.5 Interrupt is disabled then.

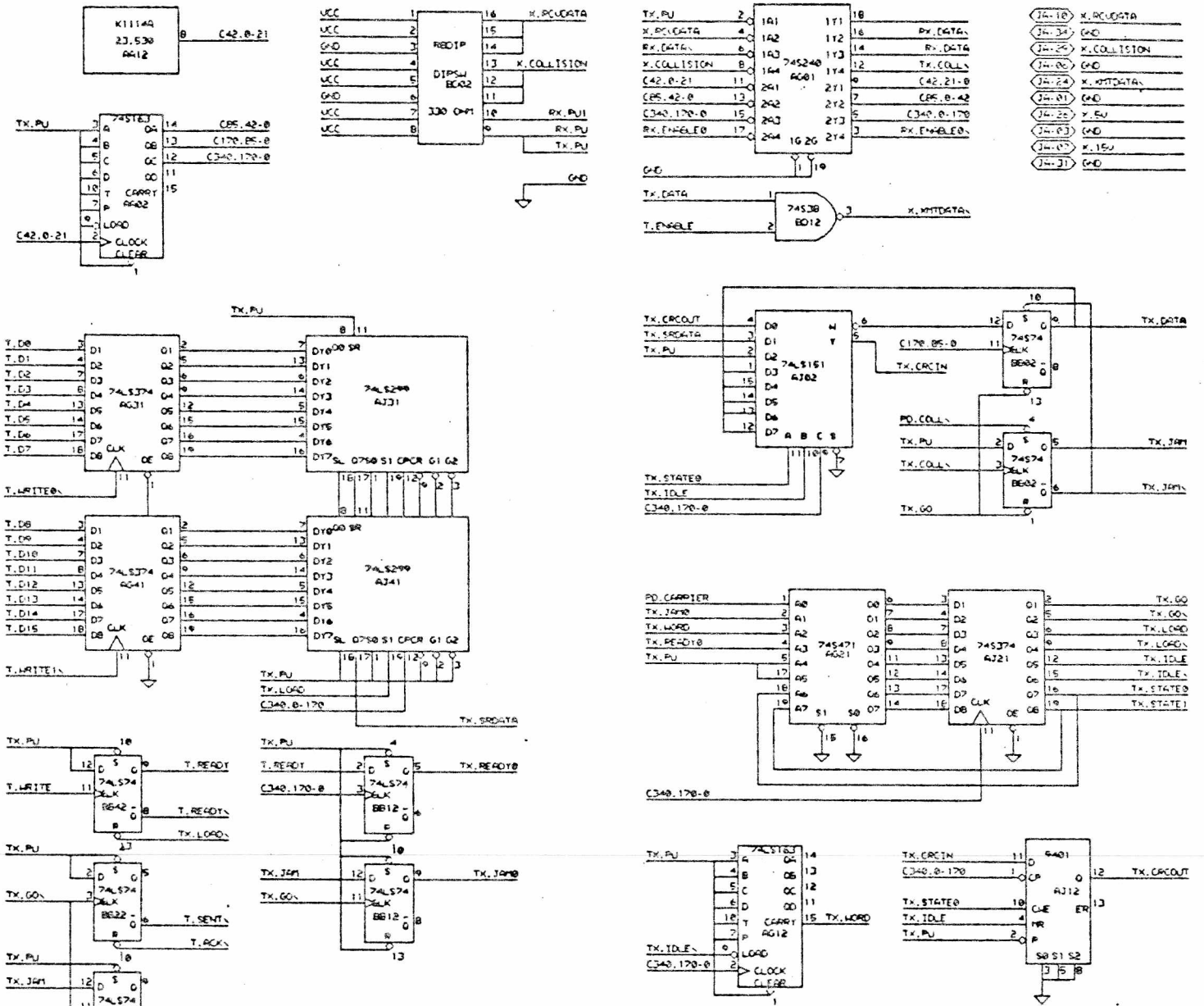
RST 6.5 (level)
RX: Message arrived (Pointer and Status was queued by Channel 1)
Reset Interrupt request
Check DMA Channel 1 current address to see how many pointers are pending.
Note that if last interrupt was asynchronous, no new message arrived.
Future Interrupts will again be handled synchronously.
If Abort then throw packet away else put on received queue.
Send message to host that packet arrived.

RST 5.5 (level)
TX: Message sent.
Check Collision Status.
Check whether DMA controller terminated transmission.
Clear request.
If Collision then if collision count < 8 then retry
else send host message about failure.
If port idle then write message header to port else put on queue.

INTR: (level)
If port idle, then Intr ← Write Interrupt request else
Intr ← EOP channel 3.
Check whether there are any pending messages for host.
Reenter idle state. (Checks on pending messages from host).



ETHERNET INTERFACE	RECEIVER	03-JUL-80 12:25	E0[SUN, AVB]

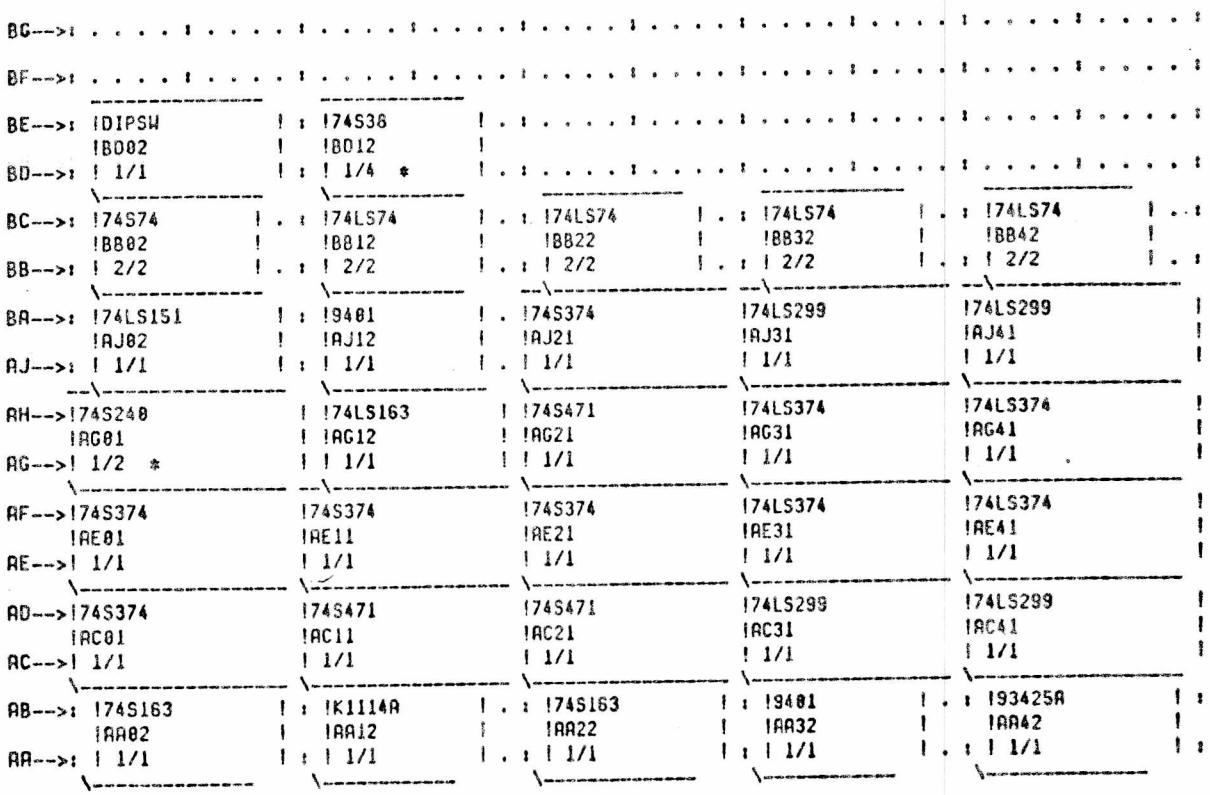


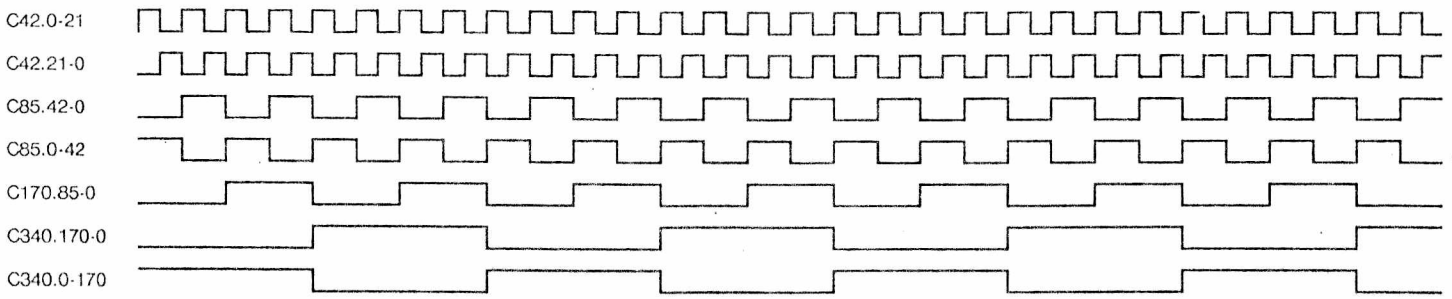
ETHERNET INTERFACE	TRANSMITTER	03-JUL-80 12:30	E1[SUN, AVB]

E.PRT[SUN,AVB] 03-JUL-88 1231

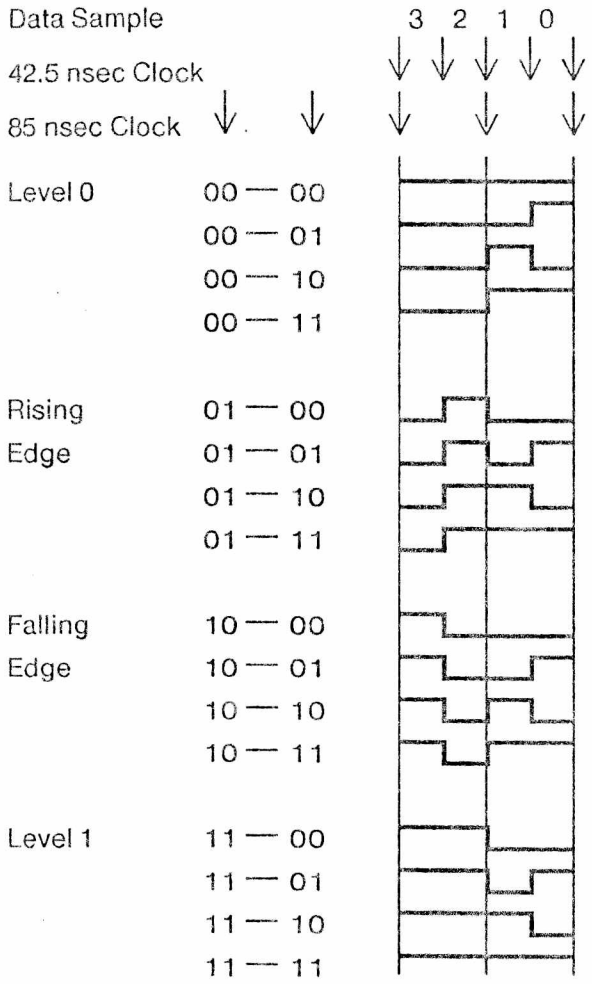
PART NUMBER	DIPTYPE	COUNT	DESCRIPTION	LOCATIONS
N/A	74S163	2		AA02, AA22
	K1114A	1	FREQ:23.530	AA12
	9401	2		AA32, AJ12
	93425A	1		AA42
	74S374	5		AC01, AE01, AE11, AE21, AJ21
	74S471	3	— 256×8	AC11, AC21, AG21
	74LS299	4		AC31, AC41, AJ31, AJ41
	74LS374	4		AE31, AE41, AG31, AG41
	74S240	1		AG01
	74LS163	1		AG12
	74LS151	1		AJ02
	74S74	1		BB02
	74LS74	4		BB12, BB22, BB32, BB42
	DIPSW	1		BD02
	74S38	1		BD12

32 ICs

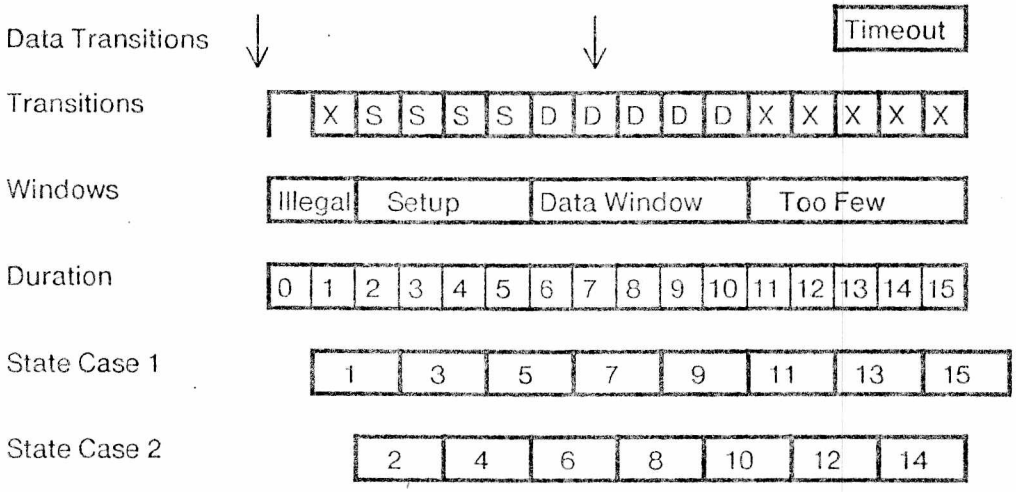

BG-->
BF-->
BE-->
!DIPSW
!B002
BD-->
! 1/1
!74S38
!B012
! 1/4 *
BC-->
!74S74
!B002
!74LS74
!B012
!74LS74
!B022
!74LS74
!B032
!74LS74
!B042
BB-->
! 2/2
! 2/2
! 2/2
! 2/2
! 2/2
BA-->
!74LS151
!AJ02
!9401
!AJ12
!74S374
!AJ21
!74LS299
!AJ31
!74LS299
!AJ41
AJ-->
! 1/1
! 1/1
! 1/1
! 1/1
! 1/1
AH-->
!74S240
!AG01
!74LS163
!AG12
!74S471
!AG21
!74LS374
!AG31
!74LS374
!AG41
AG-->
! 1/2 *
! 1/1
! 1/1
! 1/1
! 1/1
AF-->
!74S374
!AE01
!74S374
!AE11
!74S374
!AE21
!74LS374
!AE31
!74LS374
!AE41
AE-->
! 1/1
! 1/1
! 1/1
! 1/1
! 1/1
AD-->
!74S374
!AC01
!74S471
!AC11
!74S471
!AC21
!74LS299
!AC31
!74LS299
!AC41
AC-->
! 1/1
! 1/1
! 1/1
! 1/1
! 1/1
AB-->
!74S163
!AA02
!K1114A
!AA12
!74S163
!AA22
!9401
!AA32
!93425A
!AA42
AA-->
! 1/1
! 1/1
! 1/1
! 1/1
! 1/1



STANFORD CSD	Project	Reference	File	Designer	Rev	Date	Page
	SUNet	Ethernet Interface Clocks	ei.clocks.sil	John Seamons	3	30-jun-80	1

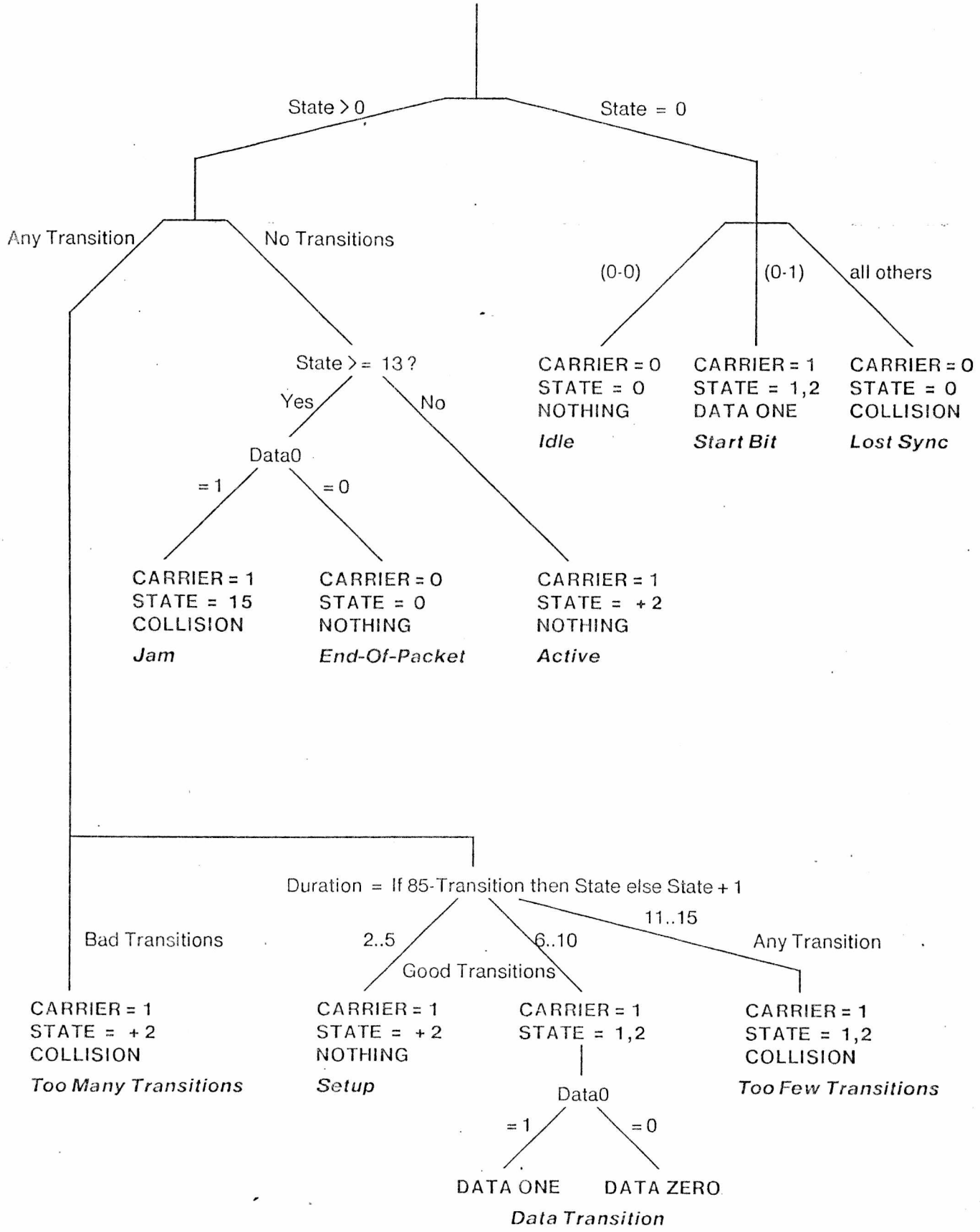


The Shift Register samples RX.Data every 42.5 nsec.
Since the Phase Decoder State Machine is clocked at 85 nsec, each 85 nsec window has 4 possible cases (Level 0, Rising Edge, Falling Edge, Level 1). Thus there are 16 possible transitions between consecutive state machine states, which are shown above.

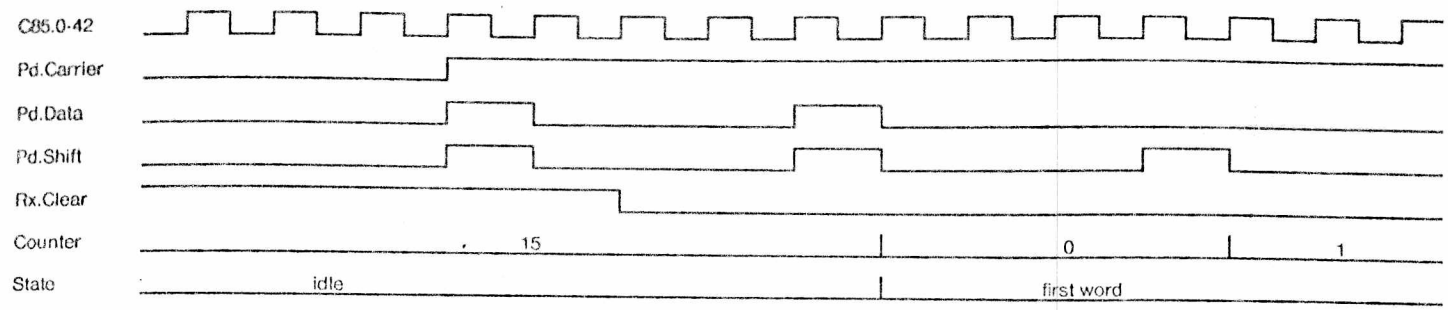

Data Transitions
Timeout
Transitions
X S S S S D D D D D X X X X X
Windows
Illegal
Setup
Data Window
Too Few
Duration
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
State Case 1
1 3 5 7 9 11 13 15
State Case 2
2 4 6 8 10 12 14

State 0 means no carrier.
State measures time between data transitions in 42.5 nsec units.
State starts at 1 if data transition falls in between 85 nsec clock, starts on 2 if data transition on 85 nsec boundary.
Duration of pulse is computed from current state, adjusted by +1 if transition falls into middle of 85 nsec cycle.

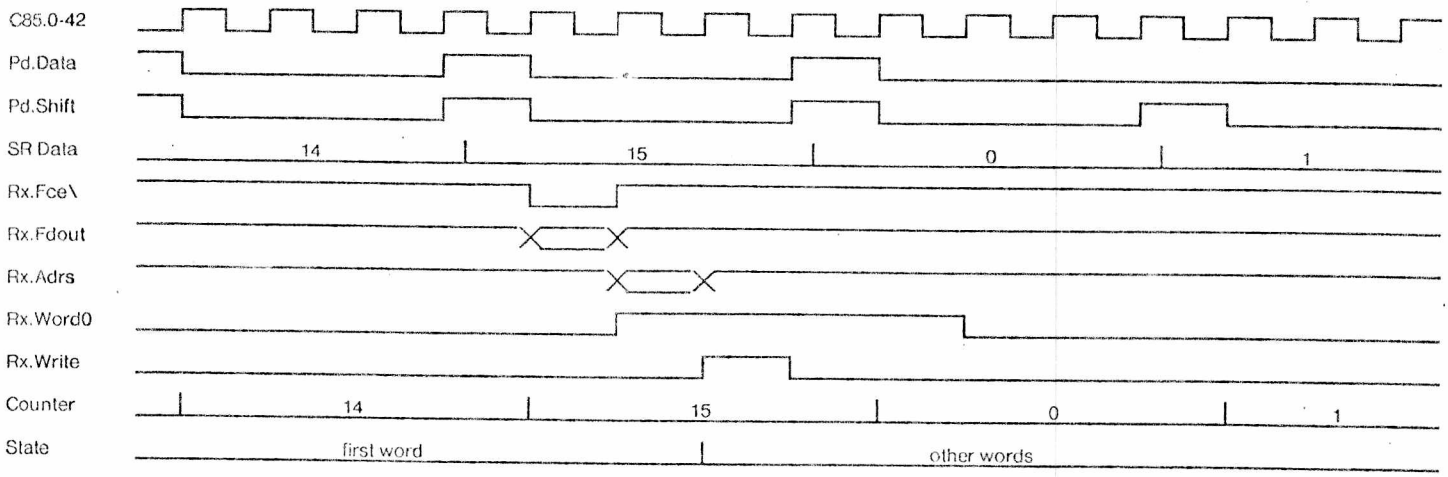
STANFORD CSD	Project SUNet	Reference Phase Decoder SR	File ei.pdsr.sil	Designer John Seamons	Rev 3	Date 30-jun-80	Page 1



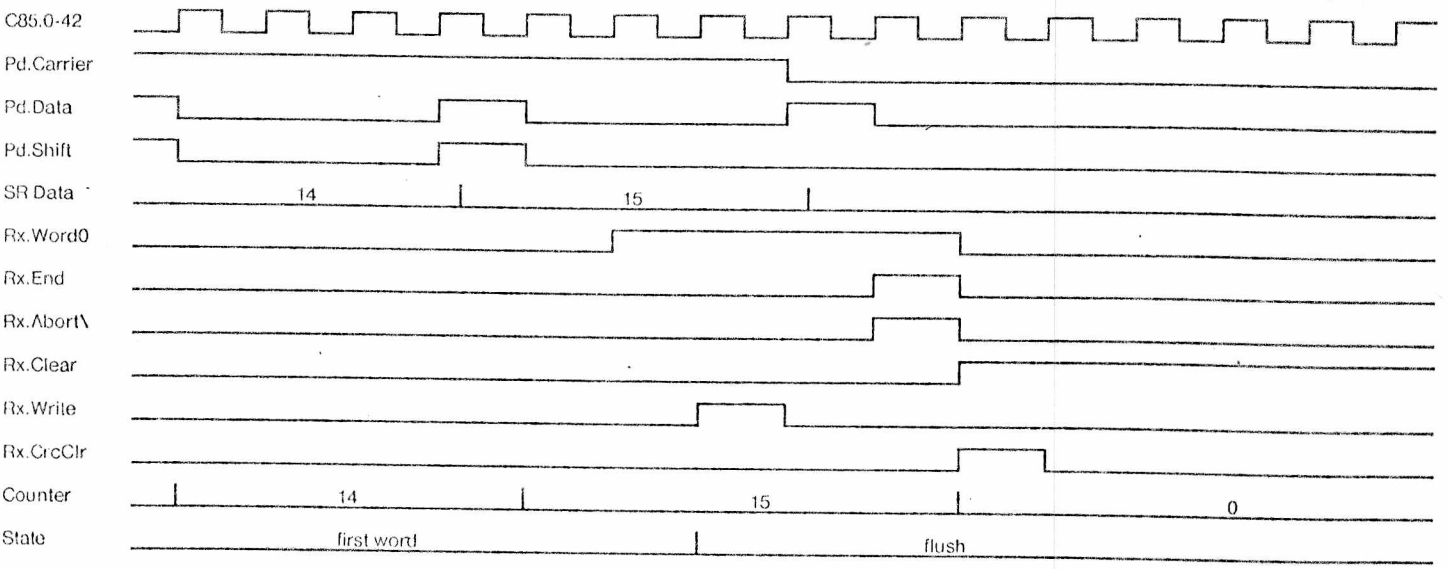
STANFORD CSD	Project	Reference	File	Designer	Rev	Date	Page
	SUNet	Phase Decoder FSM	ei.pdfsm.sil	John Seamons	3	30-jun-80	1



Start Bit

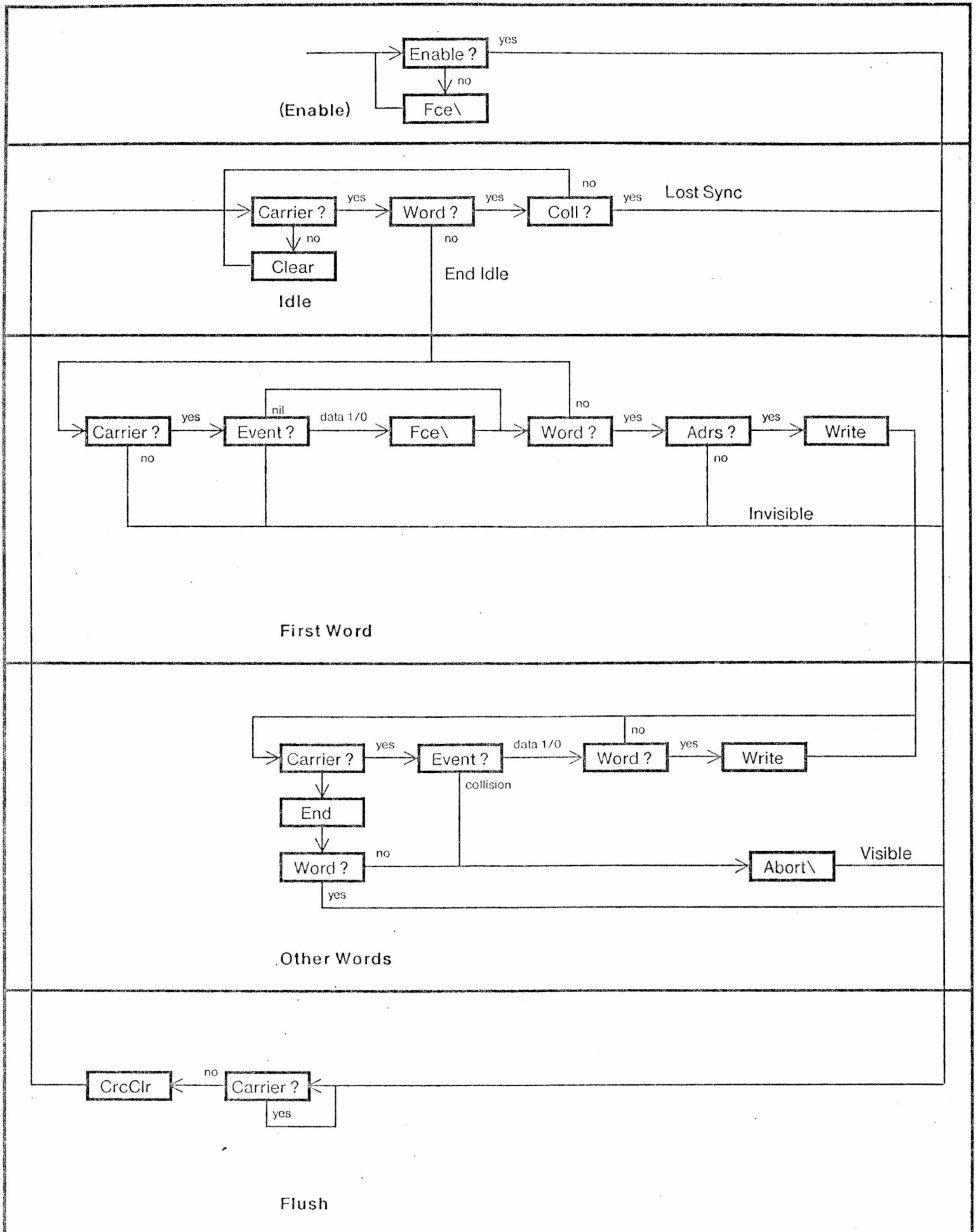


SR Full

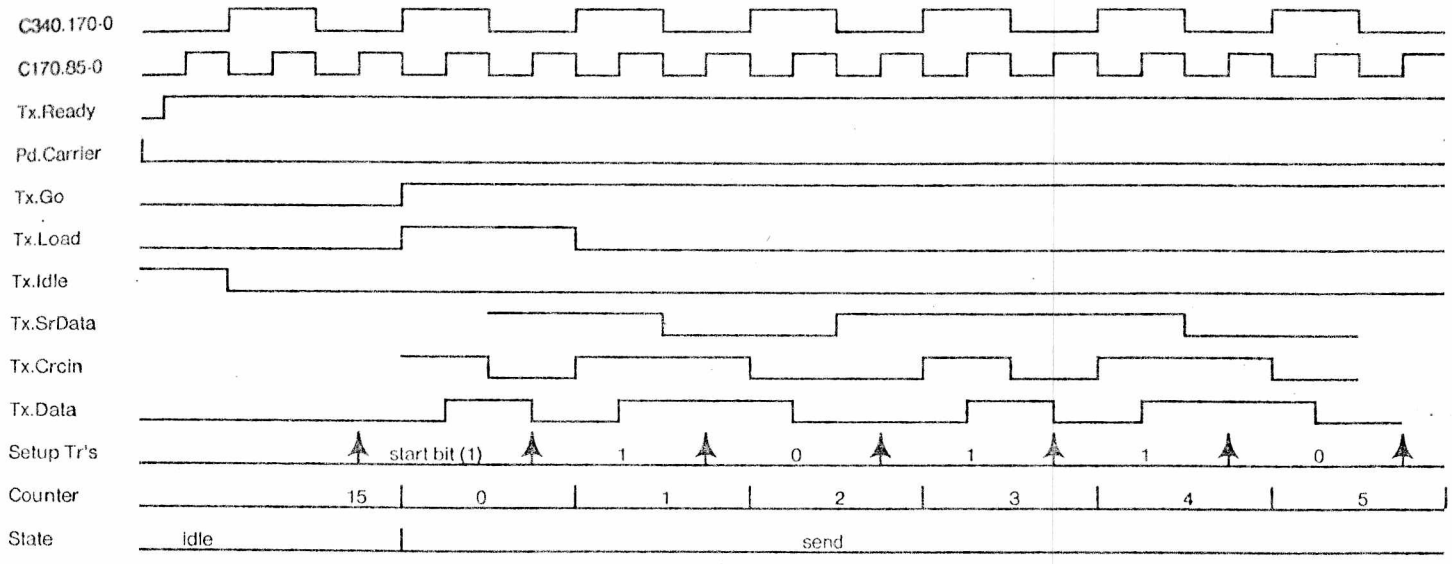


End of Packet

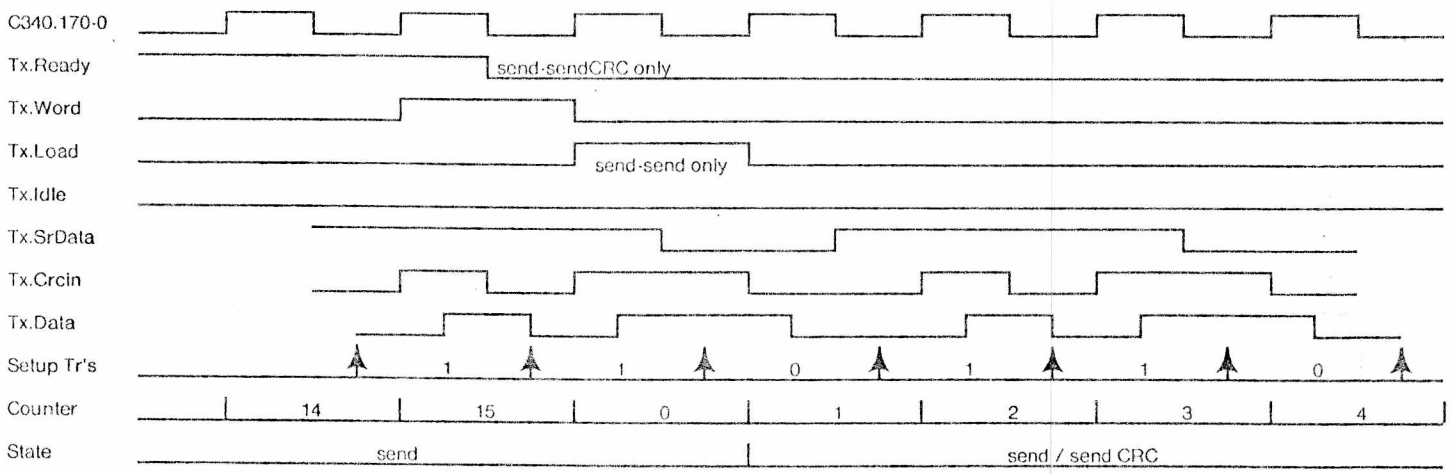
STANFORD CSD	Project SUNet	Reference Receiver Timing	File ei.rxtiming.sil	Designer John Seamons	Rev 3	Date 30-jun-80	Page 1



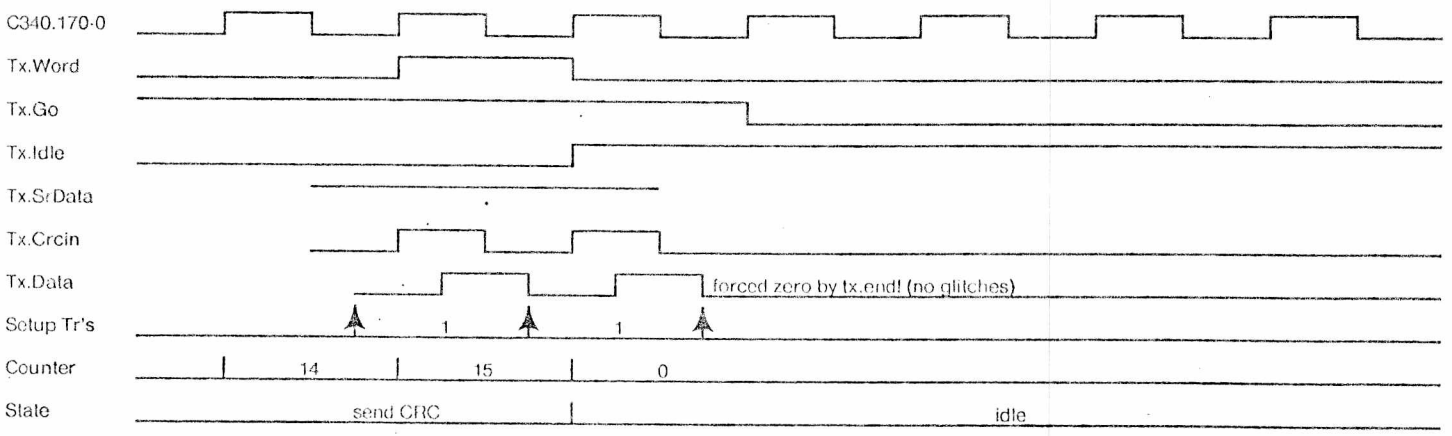
STANFORD CSD	Project SUNet	Reference Receiver FSM	File ei.rxfsm.sil	Designer John Seamons	Rev 3	Date 30-jun-80	Page 1



Idle

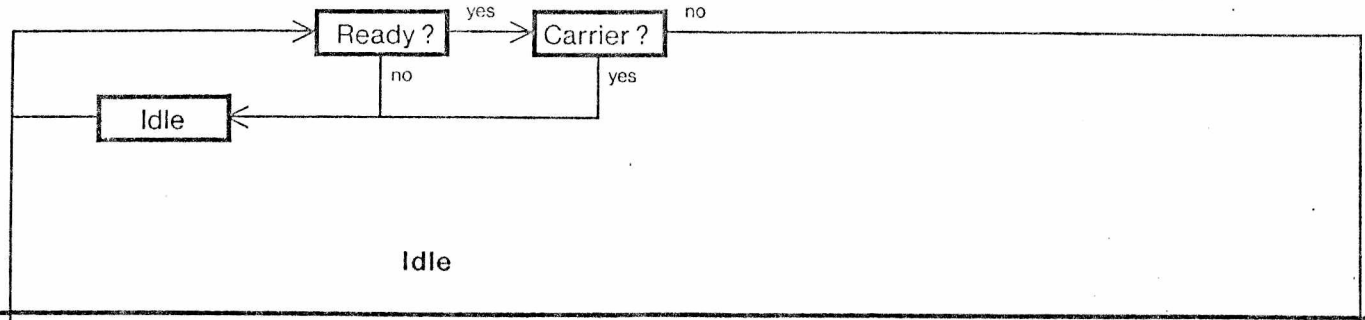


Send / Send CRC

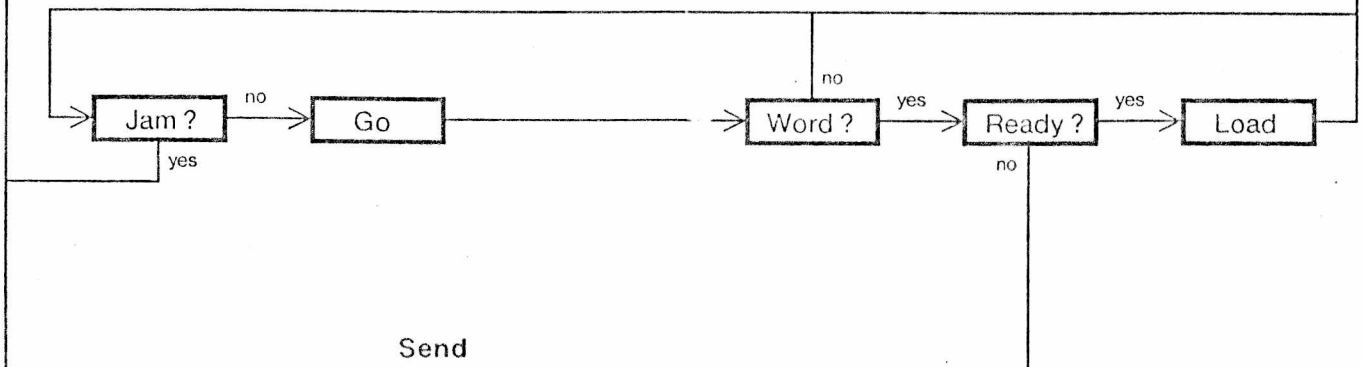


Shut Down

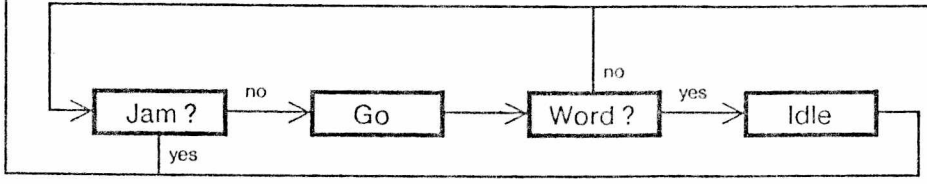
STANFORD CSD	Project	Reference	File	Designer	Rev	Date	Page
	SUNet	Transmitter Timing	ei.txtiming.sil	John Seamons	3	30-jun-80	1



Idle



Send



Send CRC

STANFORD CSD	Project SUNet	Reference Transmitter FSM	File ei.txfsm.sil	Designer John Seamons	Rev 3	Date 30-jun-80	Page 1

File under: A Date: 3 Jul 1980 Name: Andy Bechtolsheim

Name: Andy Bechtolsheim

Project: SUN Programmer: AVB

File Name: PD.SAI[SUN,AVB]

File Last Written: 17:48 3 Jul 1980

Time: 18:24 Date: 3 Jul 1980

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

data0   =[a0],
data1   =[a1],
data2   =[a2],
data3   =[a3],
state0  =[a4],
state1  =[a5],
state2  =[a6],
state3  =[a7],

$carrier=[d0],
$data   =[d1],
$shift  =[d2],
$coll_i =[d3],
$state0 =[d4],
$state1 =[d5],
$state2 =[d6],
$state3 =[d7],
```

```
state           =[((state0 + state1 + state2 + state3) div d4)],
active          =[(state≠0)],
onetransition   =[(¬data2 ∧ data0)],                    comment X0 → X1;
zerotransition  =[(data2 ∧ ¬data0)],                    comment X1 → X0;
idletransition  =[(¬data3 ∧ ¬data2 ∧ ¬data1 ∧ ¬data0)], comment 00 → 00;
goodtransition  =[(onetransition ∨ zerotransition)],
notransition    =[((¬data2 ∧ ¬data1 ∧ ¬data0)           comment X0 → 00;
                ∨ (data2 ∧ data1 ∧ data0))],            comment X1 → 11;
badtransition   =[(¬goodtransition ∧ ¬notransition)],   comment all others;
_85_transition  =[(data0 ∧ data1 ∨ ¬data0 ∧ ¬data1)],   comment in sync with 85;
duration        =[(if _85_transition then state else (state+1))],

idle            =[(¬active ∧ idletransition)],
startbit        =[(¬active ∧ onetransition)],
lostsync        =[(¬active ∧ ¬(idletransition ∨ onetransition))],
toomany         =[(active ∧ badtransition)],
setup           =[(active ∧ goodtransition ∧ (2≤duration≤5))],
datatrans       =[(active ∧ goodtransition ∧ (6≤duration≤10))],
toofew          =[(active ∧ (goodtransition ∨ badtransition) ∧ (11≤duration≤15))],
timeout         =[(active ∧ notransition ∧ (13≤state≤15))],
jam             =[(timeout ∧ data0)],
eop             =[(timeout ∧ ¬data0)],

data_zero       =[(zerotransition ∧ datatrans)],
data_one        =[((onetransition ∧ datatrans) ∨ startbit)],
collision       =[(lostsync ∨ jam ∨ toomany ∨ toofew)],
start           =[(startbit ∨ lostsync ∨ jam ∨ datatrans ∨ toofew)],
reset           =[(idle ∨ eop)],
nextstate       =[(if reset then 0
                  else if (start ∧ ¬_85_transition) then 1
                  else if (start ∧ _85_transition) then 2
                  else (state+2))];

prombegin

bit($carrier,   (nextstate≠0));
bit($data,      (collision ∨ data_one));
bit($shift,     (data_zero ∨ data_one));
bit($coll_i,    (¬collision));
bit($state0,    (nextstate land d0));
bit($state1,    (nextstate land d1));
bit($state2,    (nextstate land d2));
bit($state3,    (nextstate land d3));

if (idle ∧ (collision ∨ data_zero ∨ data_one))
        then error ("illegal idle state!");
if (collision ∧ (idle ∨ data_zero ∨ data_one))
        then error ("illegal collision state!");
if (data_zero ∧ (idle ∨ collision ∨ data_one))
        then error ("illegal data_zero state!");
if (data_one ∧ (idle ∨ collision ∨ data_zero))
        then error ("illegal data_one state!");

promend;
writeprom("pd",0);
end;
```

Listing for PROM: pd

ADRS	DATA							
0	00001000	00011111	00010011	00101111	00100011	00010011	00010011	00100011
8	00100011	00011111	00010011	00101111	00100011	00010011	00010011	00100011
16	00111001	00111001	00110011	00111001	00111001	00110011	00111001	00111001
24	00111001	00111001	00110011	00111001	00111001	00110011	00111001	00111001
32	01001001	01001001	01000011	01001001	01001001	01000011	01001001	01001001
40	01001001	01001001	01000011	01001001	01001001	01000011	01001001	01001001
48	01011001	01011001	01010011	01011001	01011001	01010011	01011001	01011001
56	01011001	01011001	01010011	01011001	01011001	01010011	01011001	01011001
64	01101001	01101001	01100011	01101001	01101001	01100011	01101001	01101001
72	01101001	01101001	01100011	01101001	01101001	01100011	01101001	01101001
80	01111001	00011111	01110011	01111001	01111001	01110011	00011101	01111001
88	01111001	00011111	01110011	01111001	01111001	01110011	00011101	01111001
96	10001001	00011111	10000011	00101111	00101101	10000011	00011101	10001001
104	10001001	00011111	10000011	00101111	00101101	10000011	00011101	10001001
112	10011001	00011111	10010011	00101111	00101101	10010011	00011101	10011001
120	10011001	00011111	10010011	00101111	00101101	10010011	00011101	10011001
128	10101001	00011111	10100011	00101111	00101101	10100011	00011101	10101001
136	10101001	00011111	10100011	00101111	00101101	10100011	00011101	10101001
144	10111001	00011111	10110011	00101111	00101101	10110011	00011101	10111001
152	10111001	00011111	10110011	00101111	00101101	10110011	00011101	10111001
160	11001001	00010011	00010011	00101111	00101101	00010011	00010011	11001001
168	11001001	00010011	00010011	00101111	00101101	00010011	00010011	11001001
176	11011001	00010011	00010011	00100011	00100011	00010011	00010011	11011001
184	11011001	00010011	00010011	00100011	00100011	00010011	00010011	11011001
192	11101001	00010011	00010011	00100011	00100011	00010011	00010011	11101001
200	11101001	00010011	00010011	00100011	00100011	00010011	00010011	11101001
208	00001000	00010011	00010011	00100011	00100011	00010011	00010011	00100011
216	00001000	00010011	00010011	00100011	00100011	00010011	00010011	00100011
224	00001000	00010011	00010011	00100011	00100011	00010011	00010011	00100011
232	00001000	00010011	00010011	00100011	00100011	00010011	00010011	00100011
240	00001000	00011001	00010011	00100011	00100011	00010011	00011001	00100011
248	00001000	00011001	00010011	00100011	00100011	00010011	00011001	00100011

```
:10000000081F132F23131323231F132F231313232B
:1000100039393339393339393933393933393968
:1000200049494349494349494943494943494958
:1000300059595359595359595953595953595948
:1000400069696369696369696963696963696938
:10005000791F737979731D79791F737979731D7994
:10006000891F832F2D831D89891F832F2D831D8930
:10007000991F932F2D931D99991F932F2D931D99A0
:10008000A91FA32F2DA31DA9A91FA32F2DA31DA910
:10009000B91FB32F2DB31DB9B91FB32F2DB31DB980
:1000A000C913132F2D1313C9C913132F2D1313C9DC
:1000B000D9131323231313D9D9131323231313D9B8
:1000C000E9131323231313E9E9131323231313E968
:1000D000081313232313132308131323131323A6
:1000E000081313232313132308131323131323396
:1000F000081913232313192308191323231319236E
:0000000000
```

File under: A Date: 3 Jul 1980 Name: Andy Bechtolsheim

Name: Andy Bechtolsheim

Project: SUN Programmer: AVB

File Name: RX.SAI[SUN,AVB]

File Last Written: 17:48 3 Jul 1980

Time: 18:05 Date: 3 Jul 1980

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

carrier         =[a0],
data            =[a1],
shift           =[a2],
word            =[a3],
adrs            =[a4],
enable          =[a5],
state0          =[a6],
state1          =[a7],

$write          =[d0],
$end            =[d1],
$abort_i        =[d2],
$clear_i        =[d3],
$crcclr         =[d4],
$fce_i          =[d5],
$state0         =[d6],
$state1         =[d7],
```

```
init            =[(¬enable)],
state           =[((state1 + state0) div d6)],
idle            =[(state=0)],
first_word      =[(state=1)],
other_words     =[(state=2)],
flush           =[(state=3)],
event           =[((shift + data) div d1)],
nil             =[(event=0)],
collision       =[(event=1)],
d_zero          =[(event=2)],
d_one           =[(event=3)],
datatr          =[(carrier ∧ (d_zero ∨ d_one ∨ nil))],
visible         =[(other_words ∧ ((¬carrier ∧ ¬word) ∨ (carrier ∧ collision)))],
invisible       =[(first_word ∧ ((¬carrier) ∨ (carrier ∧ collision) ∨
                  (datatr ∧ word ∧ ¬adrs)))],
end_idle        =[(idle ∧ carrier ∧ ¬word)],
lostsync        =[(idle ∧ carrier ∧ collision)],

nextstate       =[(
                if (       ! idle;
                (idle ∧ ¬carrier) ∨
                (idle ∧ carrier ∧ ¬(collision ∨ end_idle)) ∨
                (flush ∧ ¬carrier)
                ) then 0 else
                if (    ! first_word;
                (first_word ∧ datatr ∧ ¬word) ∨
                (end_idle)
                ) then 1 else
                if (    ! other_words;
                (first_word ∧ datatr ∧ word ∧ adrs) ∨
                (other_words ∧ datatr)
                ) then 2 else
                if (    ! flush;
                (flush ∧ carrier) ∨
                (lostsync) ∨
                (visible) ∨
                (invisible) ∨
                (other_words ∧ ¬carrier ∧ word)
                ) then 3 else
                error(cvs($$adrs)&": didn't assign any state!")
                )];

prombegin

bit($write,     (¬init ∧ ((first_word ∧ datatr ∧ word ∧ adrs) ∨
                        (other_words ∧ datatr ∧ word))));
bit($end,       (¬init ∧ ((visible) ∨ (other_words ∧ ¬carrier ∧ word))));
bit($abort_i,   ¬(init ∨ (visible)));
bit($clear_i,   ¬(init ∨ (idle ∧ ¬carrier)));
bit($crcclr,    (init ∨ (flush ∧ ¬carrier)));
bit($fce_i,     ¬(init ∨ (first_word ∧ carrier ∧ (d_zero ∨ d_one))));
bit($state0,    (init ∨ (nextstate land d0)));
bit($state1,    (init ∨ (nextstate land d1)));

promend;
writeprom("rx",0);
end;
```

Listing for PROM: rx

ADRS	DATA							
0	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
8	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
16	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
24	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
32	00100100	01101100	00100100	01101100	00100100	01101100	00100100	01101100
40	00100100	00101100	00100100	11101100	00100100	00101100	00100100	00101100
48	00100100	01101100	00100100	01101100	00100100	01101100	00100100	01101100
56	00100100	00101100	00100100	11101100	00100100	00101100	00100100	00101100
64	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
72	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
80	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
88	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
96	11101100	01101100	11101100	11101100	11101100	01001100	11101100	01001100
104	11101100	11101100	11101100	11101100	11101100	11001100	11101100	11001100
112	11101100	01101100	11101100	11101100	11101100	01001100	11101100	01001100
120	11101100	10101101	11101100	11101100	11101100	10001101	11101100	10001101
128	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
136	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
144	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
152	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
160	11101010	10101100	11101010	11101010	11101010	10101100	11101010	10101100
168	11101110	10101101	11101110	11101010	11101110	10101101	11101110	10101101
176	11101010	10101100	11101010	11101010	11101010	10101100	11101010	10101100
184	11101110	10101101	11101110	11101010	11101110	10101101	11101110	10101101
192	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
200	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
208	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
216	11010000	11010000	11010000	11010000	11010000	11010000	11010000	11010000
224	00111100	11101100	00111100	11101100	00111100	11101100	00111100	11101100
232	00111100	11101100	00111100	11101100	00111100	11101100	00111100	11101100
240	00111100	11101100	00111100	11101100	00111100	11101100	00111100	11101100
248	00111100	11101100	00111100	11101100	00111100	11101100	00111100	11101100

```
:10000000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0F0
:10001000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0E0
:10002000246C246C246C246C242C24EC242C242C90
:10003000246C246C246C246C242C24EC242C242C80
:10004000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0B0
:10005000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0A0
:10006000EC6CECECEC4CEC4CECECECECECCCECCCD0
:10007000EC6CECECEC4CEC4CECADECECEC8DEC8D7D
:10008000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D070
:10009000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D060
:1000A000EAACEAEAEAACEAACEEADEEEAEEADEEAD11
:1000B000EAACEAEAEAACEAACEEADEEEAEEADEEAD01
:1000C000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D030
:1000D000D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D020
:1000E0003CEC3CEC3CEC3CEC3CEC3CEC3CEC3CECD0
:1000F0003CEC3CEC3CEC3CEC3CEC3CEC3CEC3CECC0
:0000000000
```

File under: A Date: 3 Jul 1980 Name: Andy Bechtolsheim

Name: Andy Bechtolsheim

Project: SUN Programmer: AVB

File Name: TX.SAI[SUN,AVB]

File Last Written: 17:50 3 Jul 1980

Time: 18:25 Date: 3 Jul 1980

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

carrier         =[a0],
jam             =[a1],
word            =[a2],
ready           =[a3],
spare0          =[a4],
spare1          =[a5],
state0          =[a6],
state1          =[a7],

$go             =[d0],
$go_i           =[d1],
$load           =[d2],
$load_i         =[d3],
$idle           =[d4],
$idle_i         =[d5],
$state0         =[d6],
$state1         =[d7],
```

```
spare           =[(spare0 ∧ spare1)],
state           =[((state1 + state0) div d6)],
idle            =[(state=0)],
send            =[(state=1)],
sendcrc         =[(state=2)],
start           =[(idle ∧ ready ∧ ¬carrier)],
reload          =[(send ∧ ¬jam ∧ word ∧ ready)],
nextstate       =[(if (¬spare ∨ (state=3)) then 0
                else if (       ! idle;
                (idle ∧ (¬ready ∨ carrier)) ∨
                ((send ∨ sendcrc) ∧ jam) ∨
                (sendcrc ∧ ¬jam ∧ word)
                ) then 0 else
                if (    ! send;
                (start) ∨
                (reload) ∨
                (send ∧ ¬jam ∧ ¬word)
                ) then 1 else
                if (    ! sendcrc;
                (send ∧ ¬jam ∧ word ∧ ¬ready) ∨
                (sendcrc ∧ ¬jam ∧ ¬word)
                ) then 2 else
                error(cvs($$adrs)&": didn't assign any state!")
                )];

prombegin

bit($go,        (spare∧ ((send ∨ sendcrc) ∧ ¬jam)));
bit($go_i,      (spare∧ ¬((send ∨ sendcrc) ∧ ¬jam)));
bit($load,      (spare∧ reload));
bit($load_i,    (spare∧ ¬reload));
bit($idle,      (spare∧ (idle ∧ ¬start)));
bit($idle_i,    (spare∧ ¬(idle ∧ ¬start)));
bit($state0,    (spare∧ (nextstate land d0)));
bit($state1,    (spare∧ (nextstate land d1)));

promend;
writeprom("tx",0);
end;
```

Listing for PROM: tx

ADRS	DATA							
0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
48	00011010	00011010	00011010	00011010	00011010	00011010	00011010	00011010
56	01101010	00011010	01101010	00011010	01101010	00011010	01101010	00011010
64	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
72	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
80	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
88	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
96	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
104	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
112	01101001	01101001	00101010	00101010	10101001	10101001	00101010	00101010
120	01101001	01101001	00101010	00101010	01100101	01100101	00101010	00101010
128	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
136	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
144	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
152	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
160	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
168	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
176	10101001	10101001	00101010	00101010	00101001	00101001	00101010	00101010
184	10101001	10101001	00101010	00101010	00101001	00101001	00101010	00101010
192	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
200	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
208	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
216	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
224	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
232	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
240	00101010	00101010	00101010	00101010	00101010	00101010	00101010	00101010
248	00101010	00101010	00101010	00101010	00101010	00101010	00101010	00101010

```
:10000000000000000000000000000000000000F0
:10001000000000000000000000000000000000E0
:10002000000000000000000000000000000000D0
:10003000A1A1A1A1A1A1A1A6A1A6A1A6A1A6A1AE
:10004000000000000000000000000000000000B0
:10005000000000000000000000000000000000A0
:1000600000000000000000000000000000000090
:1000700069692A2AA9A92A2A69692A2A65652A2A70
:1000800000000000000000000000000000000070
:1000900000000000000000000000000000000060
:1000A00000000000000000000000000000000050
:1000B000A9A92A2A29292A2AA9A92A2A29292A2AA8
:1000C00000000000000000000000000000000030
:1000D00000000000000000000000000000000020
:1000E00000000000000000000000000000000010
:1000F0002A2A2A2A2A2A2A2A2A2A2A2A2A2A2A60
:00000000
```